Bài tập làm thêm

Câu 1:

**Tính đóng gói**:

<https://kipalog.com/posts/4-tinh-chat-dac-thu-cua-lap-trinh-huong-doi-tuong>

Có thể gói dữ liệu (data, ~ biến, trạng thái) và mã chương trình (code, ~ phương thức) thành một cục gọi là lớp (class) để dễ quản lí. Trong cục này thường data rất rối rắm, không tiện cho người không có trách nhiệm truy cập trực tiếp, nên thường ta sẽ che dấu data đi bằng cách dùng private, chỉ để lòi phương thức ra ngoài, chúng ta sẽ dùng phương thức get và set để mọi người có thể truy cập vào.

Ví dụ: Chúng ta có class **EncapsulationDemo** chứa thuộc tính **age.**Chúng ta không muốn các class bên ngoài tuỳ tiện truy cập **age.**Vì thế chúng ta sẽ cung cấp 2 method public để gán và lấy giá trị age.

public class EncapsulationDemo{

    private age;

    public int getAge(){

        return empAge;

    }

    public void setAge(int age){

        this.age = age;

    }

}

// File Main.java

public class Main {

    public static void main(String args[]){

         EncapsulationDemo obj = new EncapsulationDemo();

         obj.setAge(18);

         System.out.println(obj.getAge);

    }

}

**Tính trừu tượng (Abstract Class) :**

<https://blogloi.com/tinh-truu-tuong-trong-lap-trinh-huong-doi-tuong/>

– Lớp trừu tượng (Abstract Class) là lớp dùng để định nghĩa những thuộc tính và hành vi chung của những lớp khác. Nói cách khác lớp trừu tượng là lớp dùng để khai báo thuộc tính và phương thức cho các lớp khác sử dụng.

– Lớp trừu tượng không cho phép khởi tạo tham số, chỉ khai báo.

– Lớp trừu tượng được dùng như một lớp cha (base class) của các lớp có cùng bản chất. Bản chất ở đây được hiểu là kiểu, loại, nhiệm vụ của class.

– Mỗi lớp dẫn xuất (derived class – lớp con) có thể thừa kế từ một lớp trừu tượng.

– Từ khóa abstract được dùng để định nghĩa một Lớp trừu tượng.

Một số đặc điểm trong Abstract

– Tính trừu tượng được sử dụng trong các tình huống đối tượng có tồn tại nhưng thực sự nó thuộc một đối tượng con cụ thể nào đó.

– Đặc tả abstract áp dụng cho :

+ Lớp gọi là lớp trừu tượng (abstract class)

+ Hàm gọi là hàm trừu tượng (abstract method)

|  |  |
| --- | --- |
| 1  2 | abstract class <Tên Lớp>  abstract <Kiểu dữ liệu> <Tên hàm> (Đối số); // Chỉ khai báo mà không hiện thực |

Ví dụ: public abstract class Share {

abstract void drow();

void message () {

System.out.println(“I cann’t live without beging a parent “);

}

}

Class Cirle extend Shape {

Void drow(){

System.out.println(“Cirlw drow “):

}

}

Class Cone extend Shape {

Void drow(){

System.out.println(“Cone drow “):

}

}

public class runShape {

public static void main (String [] args){

Cirle cir = new Circle();

Cone con = new Cone();

cir.drow();

con.drow();

con.message();

}

}

**Tính kế thừa:**

<https://freetuts.net/tinh-ke-thua-inheritance-trong-java-1127.html>

Lớp cha có thể chia sẻ dữ liệu và phương thức cho các lớp con, các lớp con khỏi phải định nghĩa lại những logic chung, giúp chương trình ngắn gọn. Nếu lớp cha là interface, thì lớp con sẽ di truyền những contract trừu tượng từ lớp cha.

Ví dụ:

**Calculation**:

package vidu;

public class Calculation {

    protected int c;

    public void phepCong(int a, int b) {

        c = a + b;

        System.out.println("Tổng hai số = " + c);

    }

    public void phepTru(int a, int b) {

        c = a - b;

        System.out.println("Hiệu hai số = " + c);

    }

}

**MyCalculation:**

package vidu;

public class MyCalculation extends Calculation {

    public void phepNhan(int a, int b) {

        c = a \* b;

        System.out.println("Tích 2 số = " + c);

    }

    public void phepLuyThua(int a, int b) {

        c = (int) Math.pow(a, b);

        System.out.println(a + "^" + b + " = " + c);

    }

    public static void main(String[] args) {

        int a = 12, b = 2;

        MyCalculation myCalculation = new MyCalculation();

        myCalculation.phepCong(a, b);

        myCalculation.phepTru(a, b);

        myCalculation.phepNhan(a, b);

        myCalculation.phepLuyThua(a, b);

    }

}

**Tính đa hình:**

<https://freetuts.net/tinh-da-hinh-polymorphism-trong-java-1134.html>

Là tái sử dụng những đoạn mã nguồn được viết một cách tổng quát và có thể thay đổi cách ứng xử một cách linh hoạt tùy theo loại đối tượng. Tính đa hình (Polymorphism) trong Java được hiểu là trong từng trường hợp, hoàn cảnh khác nhau thì đối tượng có hình thái khác nhau tùy thuộc vào từng ngữ cảnh. Đối tượng có tính đa hình được xem như một đối tượng đặc biệt vì có lúc đối tượng này trở thành một đối tượng khác và cũng có lúc đối tượng này trở thành một đối tượng khác nữa (tùy vào từng hoàn cảnh).

Để thể hiện tính đa hình, chúng ta cần đảm bảo 2 điều kiện sau:

Các lớp phải có quan hệ kế thừa với 1 lớp cha nào đó.

Phương thức đa hình phải được ghi đè (*override*) ở lớp con. Tính đa hình chỉ được thể hiện ghi đã ghi đè lên phương thức của lớp cha.

Ví dụ: package vidu;

public class Shape {

    public void show() {

        System.out.println("Đây là phương thức show() của lớp Shape");

    }

package vidu;

public class Rectangle extends Shape {

    public void show() {

        System.out.println("Đây là phương thức show() của lớp Rectangle");

    }

}

package vidu;

public class Square extends Shape {

    public void show() {

        System.out.println("Đây là phương thức show() của lớp Square");

    }

}

package vidu;

public class Main {

    public static void main(String[] args) {

        Shape shape = new Shape();

        shape.show();   // hiển thị dòng "Đây là phương thức show() của lớp Shape"

        // bản chất của shape là Shape, nhưng vì khai báo Rectangle nên chúng ta chỉ nhìn thấy những gì mà Rectangle có

        // vì vậy sẽ chạy những hàm của Rectangle

        shape = new Rectangle();

        shape.show();   // hiển thị dòng "Đây là phương thức show() của lớp Rectangle"

        // tương tự lúc này shape sẽ đóng vai trò là 1 Square

        shape = new Square();

        shape.show();   // hiển thị dòng "Đây là phương thức show() của lớp Square"

    }

}

Câu 2: Lợi ích của việc sử dụng đối tượng

Có tính tái sử dụng cao

Dễ quản lí code và tìm lại code dễ dàng

Có tính bảo mật

Câu 3: Khi nào thì sử dụng Abstract class:

Link: <https://loda.me/ban-thuc-su-da-biet-khi-nao-dung-interface-khi-nao-dung-abstract-loda1557932040381/>

Abstract (trừu tượng) nghĩa là một cái gì đó không hoàn toàn cụ thể, nó chỉ là một ý tưởng hoặc ý chính của một cái gì đó mà không có bản triển khai cụ thể. Vì vậy Class abstract chỉ là một cấu trúc hoặc hướng dẫn được tạo cho các class cụ thể khác.

Chúng ta có thể nói rằng một class abstract là linh hồn của một class cụ thể, và rõ ràng một cơ thể (class) không thể có hai linh hồn. Đây cũng là lý do Java không hỗ trợ nhiều kế thừa cho các class abstract.

Câu 4: Khi nào sử dụng Interface:

Interface (Giao diện) là một hình thức, giống như một hợp đồng, nó không thể tự làm bất cứ điều gì. Nhưng khi có một class ký kết hợp đồng (implement Interface) này, thì class đó phải tuân theo hợp đồng này.

Trong Interface, chúng tôi định nghĩa các hành vi của một class sẽ thực hiện. Một class có thể có một số cách hành vi khác nhau, cũng giống như nó có thể ký kết được với nhiều hợp đồng khác nhau. Đó cũng là lý do tại sao Java cho phép implement nhiều Interface.

Câu 5: Overriding (ghi đè phương thức) là gì:

Danh sách các đối số phải giống với phương thức được ghi đè.

Kiểu trả về phải giống với kiểu trả về được khai báo trong phương thức được ghi đè của lớp cha.

Mức truy cập của phương thức ghi đè phương thức ở lớp cha không được nhỏ hơn phương thức đó trong lớp cha. Ví dụ: phương thức trong lớp cha được khai báo là public thì phương thức ghi đè phương thức đó trong lớp con không thể có phạm vi truy cập là private hoặc protected.

Một phương thức được khai báo là final hoặc static thì phương thức không thể được ghi đè.

Các hàm tạo của lớp cha không thể được ghi đè.

Ví dụ: class1.java

package viduoverriding;

public class Subclass extends Superclass {

@Override

public void hienThi() {

System.out.println("Đây là phương thức hiển thị của lớp con Subclass.");

}

Class2:

package viduoverriding;

public class TestOverriding {

public static void main(String[] args) {

// khai báo đối tượng của lớp Superclass

Superclass superclass = new Superclass();

// khai báo đối tượng có bản chất là Superclass

// nhưng đóng vai trò là 1 Subclass

// vì vậy sẽ chạy những hàm của Subclass

Superclass subclass = new Subclass();

// gọi phương thức hienThi() của lớp cha

superclass.hienThi();

// gọi phương thức hienThi() của lớp con

subclass.hienThi();

}

}

Câu 6: Overloading ( Nạp chồng) là gì?

Khác nhau về số đối số truyền vào và các đối số có cùng kiểu dữ liệu.

Có cùng số đối số truyền vào và các đối số không có cùng kiểu dữ liệu.

Khác nhau trình tự kiểu dữ liệu của các đối số.

Chúng ta có 2 cách để thực hiện nạp chồng phương thức đó là thay đổi số đối số truyền vào, thay đổi kiểu dữ liệu của các đối số và thay đổi trình tự của các đối số đó.

Ví dụ: **\* Thay đổi số đối số truyền vào**

Class1:

package viduoverloading;

public class PhepCongHaiSo {

public int add(int a, int b) {

return a + b;

}

public int add(int a, int b, int c) {

return a + b + c;

}

}

Class2:

package viduoverloading;

public class TestOverloading {

public static void main(String[] args) {

PhepCongHaiSo test = new PhepCongHaiSo();

System.out.println("Tổng = " + test.add(11, 12));

System.out.println("Tổng = " + test.add(11, 12, 13));

}

}

**\* Thay đổi kiểu dữ liệu của đối số truyền vào:**

DisplayOverloading.java

package viduoverloading;

public class PhepCongHaiSo {

public int add(int a, int b) {

return a + b;

}

public float add(float a, float b) {

return a + b;

}

}

TestOverloading.java

package viduoverloading

public class TestOverloading {

    public static void main(String[] args) {

        PhepCongHaiSo test = new PhepCongHaiSo();

        System.out.println("Tổng = " + test.add(11, 12));

        System.out.println("Tổng = " + test.add(11.1f, 12.2f));

    }

}

**\*Thay đổi trình tự kiểu dữ liệu của các đối số**

Class1:

package viduoverloading;

public class DisplayOverloading {

public void hienThi(int a, char b) {

System.out.println(a + "\t" + b);

}

public void hienThi(char b, int a) {

System.out.println(b + "\t" + a);

}

}

Class2:

package viduoverloading;

public class TestOverloading {

public static void main(String[] args) {

DisplayOverloading displayOverloading = new DisplayOverloading();

displayOverloading.hienThi(5, 'a');

displayOverloading.hienThi('a', 5);

}

}

**Câu 7: Các access modifier trong java, khi dùng nó thì quyền truy cập sẽ như thế nào?**

Default: Khi một phương thức hoặc biến được khai báo là default thì chỉ có các class thuộc cùng package với nó mới có thể truy cập..

Private: Khi một phương thức hoặc biến được khai báo là private nó sẽ không thể truy cập từ class khác, kể cả các class cùng source file hay các class con.

Public: Khi một phương thức hoặc biến được khai báo là public, có nghĩa là tất cả các class khác, kể cả các class không thuộc cùng package đều có thể truy cập.

Protected: Protected modifier khá giống với default modifier, nó hạn chế khả năng truy cập trong cùng 1 package, tuy nhiên với protected modifier thì nó còn cho phép truy cập từ các class con kể cả khi class con không nằm cùng package với class cha. (truy cập theo trường hợp thừa kế)